

PR



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/520,405	03/08/2000	Michael G. Martinek	307.029US1/PA0390	1300

7590 03/20/2003

Mark A Litman & Associates PA
York Business Center Suite 205
3209 West 76th St
Edina, MN 55435

EXAMINER

ASHBURN, STEVEN L

ART UNIT	PAPER NUMBER
----------	--------------

3714

DATE MAILED: 03/20/2003

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary

Application No.

09/520,405

Applicant(s)

MARTINEK ET AL.

Examiner

Steven Ashburn

Art Unit

3714

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133).
- Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

- 1) ☒ Responsive to communication(s) filed on 09 December 2002.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

- 4) ☒ Claim(s) 1-57 is/are pending in the application.
- 4a) Of the above claim(s) _____ is/are withdrawn from consideration.
- 5) ☐ Claim(s) _____ is/are allowed.
- 6) ☒ Claim(s) 1-57 is/are rejected.
- 7) ☐ Claim(s) _____ is/are objected to.
- 8) ☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☐ The drawing(s) filed on _____ is/are: a) ☐ accepted or b) ☐ objected to by the Examiner.
- Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
- 11) ☒ The proposed drawing correction filed on 09 December 2002 is: a) ☒ approved b) ☐ disapproved by the Examiner.
- If approved, corrected drawings are required in reply to this Office action.
- 12) ☐ The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. §§ 119 and 120

- 13) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some * c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
 - ☐ Certified copies of the priority documents have been received in Application No. _____.
 - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
- * See the attached detailed Office action for a list of the certified copies not received.
- 14) ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).
- a) ☐ The translation of the foreign language provisional application has been received.
- 15) ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO-1449) Paper No(s) _____.
- 4) ☐ Interview Summary (PTO-413) Paper No(s) _____.
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: _____.

MARK SAGER
PRIMARY EXAMINER

DETAILED ACTION

Drawings

The proposed drawing correction and/or the proposed substitute sheets of drawings, filed on December 9, 2002 have been accepted. A proper drawing correction or corrected drawings are required in reply to the Office action to avoid abandonment of the application. The correction to the drawings will not be held in abeyance.

Claim Rejections - 35 USC § 112

The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

Claim 57 rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. There is insufficient antecedent basis for the limitations in the claim because the dependant claim does not list a prior claim. For examination, the examiner assumes claim 57 is depends upon claim 1.

Claim Rejections - 35 USC § 103

Claims 1-3, 11-13, 16, 19, 29, 31, 34-37, 39-43, 48, 50-54 are rejected under 35 U.S.C. 103(a) as being unpatentable over Bunnell, U.S. Patent 6,075,939 (Jun. 13, 2000).

Bunnell discloses a universal operating system with a modular kernel customizable to perform specialized applications. *See fig. 1, 2; 2:66-3:3*. The system's modularity allows it to be customized to support a range of applications ranging from small, embedded-type systems to large, network-capable instantiations. *See col. 2:66-4:2*. As a result, a developer may incorporate whatever optional kernel components desired, including incorporating application programs to maximize the system's resources by reducing computational overhead. *See id.*

In regards to claims 1, 13, 16, 19, 29, 39, 43 and 48: *Bunnell* describes the following features of the claimed invention:

- a) A computerized controller having a processor, memory and nonvolatile storage. *See fig. 2.*
- b) System handler, executed by the operating system kernel, operable to dynamically link with at least on a program object. *See fig. 1; 3:21-58.*
- c) Application Program Interface (API) callable from the program object. *See col. 9:51-54.*
- d) Operating system kernel that executes the system handler application. *See fig. 2; 3:21-58.*

Hence, *Bunnell* describes all the features of the claimed operating system except a “game” controller operable to control a “wagering game” by processing “gaming” program objects. Regardless of the deficiencies, applying the operating system described by *Bunnell* in a gaming device would have been obvious to an artisan.

Gaming systems are specialized applications that employ processors for executing wagering programs. Analogous controllers are widely used to control commercial devices having specialized functions including, for example, point-of-sale and security systems. *See, e.g., col. 8:16-19.* A gaming artisan would possess knowledge of these analogous processors for controlling specialized applications because they are pertinent to the problems of controlling specialized, commercial systems. Thus, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the operating system taught by *Bunnell* to apply in a “game” controller operable to control a “wagering game” by processing “gaming” program objects to provide an improved wagering game having with a modern operating system having the features of stability, configurability, minimal hardware requirements, low system management needs, high performance, real time support, low operating overhead, and be adaptability. *See col. 1:26-2:63.*

Art Unit: 3714

In regards to claims 2, 31, and 42, *Bunnell* additionally teaches a system handler comprising a plurality of device handlers. *See figs 1-5.*

In regards to claim 3, *Bunnell* additionally teaches a system handler unloading, loading and executing program objects. *See fig. 1-5; See 3:43-59; 9:50-54; 11:20-12:15.*

In regards to claims 6 and 29, *Bunnell* describes all the features of the claims except using a pc-based system. Regardless, it was notoriously well known at the time of the invention to employ PC-based systems in custom applications because of their cost, availability, flexibility, compatibility and broad commercial support. Furthermore, it was well known in the gaming art at the time of the invention to employ personal computers as game controllers. Thus, it would have been obvious to a gaming artisan at the time of the invention to modify the operating system suggested by *Bunnell* to implement gaming systems with IBM PC-compatible controllers to decrease the cost and increase the supportability of a gaming device.

In regards to claim 11, 35 and 40: *Bunnell* additionally teaches a system handler comprising a plurality of API's with a library of functions callable from program objects. *See fig. 4; col. 3:43-59, 9:50-54, 11:20-12:15.*

In regards to claim 12, 34 and 41: *Bunnell* additionally teaches a system handler managing an event queue determining the order of the device handlers. *See fig. 1, 2; 3:39-42; 5:26-46, 6:26-47.*

In regards to claims 36 and 37, *Bunnell* discloses a customizable operating system with an event queue, however it does not describes the specifics of the queue management. Thus, *Bunnell* describes all

Art Unit: 3714

the features of the instant claims except the event queue queuing on first-come, first-server basis (*claim 36*) and the event queue queuing using more than one criteria (*claim 37*). Regardless of the deficiencies, the features were known in the art at the time of the invention and would have been obvious to an artisan.

Event queues are a basic function of an operating system for managing the system's response to events. It is fundamental programming technique to arrange the event response protocol to respond to events in the most effective order. For example, in some cases, event queues are simply organized on a first-serve/first-serve basis. In other cases, the some events are more critical than others. Thus, the event queues are simply organized on a priority basis. In still other cases, events are handled on both a priority and first-come/first-serve basis. It would be a matter of design choice as to which manner the event queue managed events.

Thus, it would have been obvious to an artisan at the time of the invention to modify the gaming operating system suggested by *Bunnell* to manage to event queuing on first-come, first-server basis or using more than one criteria to manage the event queue's priorities to respond to common events, such as button presses, on a first-come/first-serve basis while responding to critical events, such as security faults, immediately based on a higher priority.

In regards to claims 50 and 51, *Bunnell* additionally describes a network-capable operating system. *See col. 3:1-3*. Furthermore, it is notoriously well known in the art of gaming to link gaming devices to a networked "on-line" system to, for example, monitor the devices, exchange data with a central server, track player's activities, allow cashless gaming and operate progressive meter games. Thus, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the operating system describe by *Bunnell*, wherein the operating system is customizable for specialized applications such as gaming, to operate within an networked online system.

In regards to claims 52, the operating system described by *Bunnell* teaches or suggests every feature of the claim except dynamically linking an API from a system from the system handler. Regardless, it is notoriously well known to dynamically link functions or data to a program during execution to reduce the resources used by a system by only linking when the object is required by a program or to provide for the addition or deletion of new components. Furthermore, if the did not dynamically link than the system would be static and require recompilation whenever a new component was added or removed and thereby provide. Thus, in this case, wherein the system is an embedded controller in a specialized application such as gaming machine, it would be obvious to an artisan at the time of the invention to modify the operating system described by *Bunnell*, wherein the operating system is customizable for specialized applications such as gaming, to dynamically link an API from a system from the system handler to provide a more robust, scalable, flexible operating system.

In regards to claims 53 and 54, the operating system described by *Bunnell* teaches or suggests every feature of the claim except having instructions, when executed, operable to dynamically link an API to a program object. Regardless, it is notoriously well known to dynamically link functions or data to a program during execution to reduce the resources used by a system by only linking when the object is required by a program. Thus, it would have been obvious to an artisan at the time of the invention to modify the operating system described by *Bunnell*, wherein the operating system is customizable for specialized applications such as gaming, to dynamically link an API to a program object to reduce the resources used by a system and thereby increase efficiency.

Claims 13, 14, 18-27 and 49 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of Fullerton, U.S. Patent 4,607, 844 (Aug. 26, 1986).

In regards to claims 13, 19, 24, 25, 27 *Bunnell* additionally teaches the following features:

Art Unit: 3714

- a. Causing a system handler application to load and execute program objects. *See fig. 2-5.* More specifically, a primary purpose of a system handler serves in an operating system to load and execute program objects. Hence, although *Bunnell* does not explicitly describe the feature, it is implicit in *Bunnell's* operating system.
- b. Causing a loaded program object to call-up a library of functions provided by the system handler. *See id.* More specifically, a fundamental purpose of a system handler in an operating system is to simplify calls from programs to storage devices to access resources such as function libraries. Hence, although *Bunnell* does not explicitly describe the feature, it is implicit in *Bunnell's* operating system.
- c. Load a first program object from the library, execute the first program object, unload the first program object and load the second program object. *See id.* A fundamental purpose of an operating system is to manage the device's physical memory. In executing a program, the operating system is required to place program objects in memory to execute them, and subsequently remove the program object to swap-in a second program object for execution. Hence, the claimed feature is implicit in *Bunnell's* operating system.
- d. Store data variables in storage such that a second program object in the library later loaded can access the data variables in storage. *See fig. 2-5.*

However, *Bunnell* does not describe storing data variables in non-volatile memory

Fullerton discloses a gaming machine that stores data variables in non-volatile storage to increase security due to tampering or loss of power. *See col. 1:5-2:41.* In view of *Fullerton*, it would have been obvious to an artisan at the time of the invention to modify the operating system describe by *Bunnell*, wherein the operating system is customizable for specialized applications such as gaming, to store data variables in non-volatile storage to increase security.

Art Unit: 3714

In regards to claim 23, it is implicit in the operating system describe by *Bunnell* that instructions are operable when executed to cause a computer to provide functions through an API that comprises part of the system handler application because a fundamental function of a system handler is to provide an API to simplify functions calls by applications to the kernel.

Claims 15 and 49 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of *Fullerton*, as applied to claims 13 above, in further view Pascal et al., U.S. Patent 5,791,851 (Oct. 26, 1999).

Bunnell additionally describes a kernel module providing security management for the system. *See col. 16-29*. The module monitors applications for access control and reporting. *See id.* Hence, *Bunnell* generally suggests monitoring application modules to ensure the security of the systems.

The gaming device suggested by either *Bunnell* with *Fullerton* describes all the features of the claimed subject matter except executing a corresponding callback function upon alteration of variable data in non-volatile storage. Regardless of the deficiency, this feature would have been obvious to an artisan in view of *Pascal*.

Pascal discloses an analogous operating system for a gaming device wherein callbacks are employed to communicate information between application modules upon the occurrence of certain events. *See 1:44-2:30*. In general, callback routines are used in state-based machines to communicate data between independent modules upon the occurrence of predetermined events. *See col. 6:25-45*. *Pascal* describes using callback to enhance to robustness of a gaming device under fault conditions to protect data that may affect the outcome of a game payout. *See col. 2:25-30*.

In view of *Pascal*, it would have been obvious to an artisan at the time of the invention to modify the customized gaming operating system suggested by the combination of *Bunnell* with *Fullerton*, wherein the system enhance security by monitoring application modules, to execute a callback function

Art Unit: 3714

corresponding to a change in game data stored in non-volatile memory to enhance the security of the gaming device by monitoring changes in data that might affect the outcome of the game payout and thereby provide a more secure gaming device that is resistant to errors caused by losses in power or tampering.

In regards to claim 15, *Bunnell* additionally teaches handling events via a system handler application. *See col. 6:24-7:8:14.*

In regards to claim 21, it is implicit in *Bunnell* that instructions operable when executed to cause a computer to manage events via the system handler application because managing events is a primary function of a system handler in an operating system.

Claims 28-33 are rejected under 35 U.S.C. 103(a) as being unpatentable over Houriet, Jr. et al., U.S. Patent 5,575,717 (Nov. 19, 1996) in view of Mitchell et al., U.S. Patent 5,872,973 (Feb. 16, 1999).

In regards to claim 28, *Houriet* discloses a system for allowing a game operator to individually tailor parameters of a video game machine, ^{to} provides a plurality of user interactive video screen displays for selecting the parameters. The parameters include the selection of menu choices of video games from a library of video games that are available for play on the machine. *See abstract.* In particular regards to the claim, *Houriet* teaches a gaming device incorporating a program containing a large number of games wherein a mode selector allows selection of the game to be executed by the device. *See col. 1:61-2:36.* Furthermore, it is inherent that electronic controller that operates the electronic gaming system includes an operating system for controlling the hardware and software that comprises the game. Thus, *Houriet* teaches a gaming machine comprising an operating system and a plurality of game programs describing a

Art Unit: 3714

game personality is a selected mode. However, the reference does not describe a plurality of shared objects wherein each shared object describes the game personalities. Regardless of the deficiency, this feature would have been obvious to an artisan.

Mitchell a system for creating named relations between classes in a dynamic object-oriented programming environment. The objects dynamically bind to the class interfaces of the classes being related. The relationships can be programmatically attached by name to object instances during program execution. Because these relationships are stored in a resource and are dynamically bound by name to the objects, they can be created and modified without requiring the source code of the objects being associated to be changed. This eliminates hard coded dependencies between objects that impede reuse of the objects in other contexts. In particular regards to the claim, *Mitchell* teaches that it was known in the art to structure programs into sharable modules that are linked to form complex programs. *See col. 1:24-2:57, 4:30-5:19*. As a result, program development becomes simpler and less costly because modules can be reused instead of recreated each time. *See id.* Thus, it was known at the time of the invention to form a program from a plurality of shared objects wherein each describes features of the program.

In view of *Mitchell*, it would have been obvious to an artisan at the time of the invention to modify the gaming system described by *Houriet*, wherein the device contains a plurality of selectable game modes having describing a game personality is a selected mode, to add the feature of shared objects wherein each shared object describes the game personalities. As taught by *Mitchell*, the modification would enhance the system by making program development simpler and less costly because modules can be reused instead of recreated. *See id.*

In regards to claim 29, *Mitchell* additionally describes an operating system based on an IBM-PC. *See col. 4:30-49*.

Art Unit: 3714

In regards to claim 30, *Houri* describes a gaming device wherein an electronic controller responds to events such as user inputs to a touchscreen display. *See fig. 4; col. 3:33-57*. Hence, the reference includes an operating system with software capable of detecting and responding to events. However, *Houri* does not explicitly describe a system handler. Regardless of the deficiency, this feature would have been obvious to an artisan.

A system handler is a virtual device created in software that provides an interface between application programs and machine-level devices in order to automate the task of exchanging data and thereby reduce to complexity and effort required to write applications by allowing higher-level applications to call upon lower-level services. System handlers typically respond to events detected by the system. For example, in a gaming device, an event requiring a response would allow a security application to receive a signal from hardware indicating the opening of access panel on the housing of the gaming device causing the event to be logged in storage and activating an indicator. Thus, it is known in the art to employ system handlers operating systems for gaming devices.

In regards to claim 31, as described above, *Houri* describes all the features of the claims except a device handler. Regardless of the deficiency, this feature would have been obvious to an artisan.

A device handler is a virtual device created in software that provides an interface between application programs and hardware devices in order to automate the task of exchanging data and thereby reduce to complexity and effort required to write applications. For example, in a gaming device, an event requiring a response would include opening an access panel to allow the event to be logged in storage and activate an indicator. Thus, it is known in the art to employ device handlers in operating systems for gaming devices.

Consequently, it would have been obvious to an artisan at the time of the invention to modify the gaming system described by *Houri*, wherein a operating system controls a electronic processor, to add

Art Unit: 3714

the feature of a system handler to automate the task of exchanging data and thereby reduce to complexity and effort required to write applications.

In regards to claim 32, as described above, *Houriet* describes all the features of the claims except an event queue. Regardless of the deficiency, this feature would have been obvious to an artisan.

Scheduling is a task performed by a system handler to allow a computer to respond to events based on priority. Typically, a event queue is generated that schedules initiation of routines associated with the events. For example, the handling events such as touch-screen inputs would commonly be a higher priority than a door-open because of the faster response time required to by the event. Thus, it is known in the art to employ event queue in operating systems for gaming devices. Consequently, it would have been obvious to an artisan at the time of the invention to modify the gaming system described by *Houriet*, wherein a operating system controls a electronic processor, to add the feature of a event queue to allow the system to respond to events based on priority.

In regards to claims 33, *Mitchell* additionally describes a plurality of API callable functions. *See col. 1:24-2:57.*

In regards to claims 36 and 37, *Bunnell* discloses an customizable operating system with an event queue, however it does not describes the specifics of the queue management. Thus, *Bunnell* describes all the features of the instant claims except the event queue queuing on first-come, first-server basis (*claim 36*) and the event queue queuing using more than one criteria (*claim 37*). Regardless of the deficiencies, the features were known in the art at the time of the invention and would have been obvious to an artisan.

Event queues are a basic function of an operating system for managing the system's response to events. It is fundamental programming technique to arrange the event response protocol to respond to

Art Unit: 3714

events in the most effective order. For example, in some cases, event queues are simply organized on a first-serve/first-serve basis. In other cases, the some events are more critical than others. Thus, the event queues are simply organized on a priority basis. In still other cases, events are handled on both a priority and first-come/first-serve basis. It would be a matter of design choice as to which manner the event queue managed events.

Thus, it would have been obvious to an artisan at the time of the invention to modify the gaming operating system suggested by *Bunnell* to manage to event queuing on first-come, first-server basis or using more than one criteria to manage the event queue's priorities to respond to common events, such as button presses, on a first-come/first-serve basis while responding to critical events, such as security faults, immediately based on a higher priority.

Claims 7, 8, and 14 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of David A. Rusling, *The Linux Kernel*, <<http://www.tldp.org/LDP/tlk/tlk.htm>> (1999) (hereinafter "*Rusling*").

In regards to claim 7, *Bunnell* teaches all the features of the claim except using a LINUX operating system kernel. Regardless of the deficiency, this feature would have been obvious to an artisan.

Rusling describes the features of the LINUX operating system. The system is a well-known, freely distributed, PC-compatible operating system derived from UNIX. Analogously to *Bunnell*, the LINUX kernel is scalable and customizable for specialized applications. A gaming artisan at the time of the invention would possess knowledge of the fundamental features and relative advantages of the various operating systems, including LINUX. Thus, it would have been obvious to gaming artisan at the time of the invention to modify *Bunnell*'s operating system to be based upon a LINUX foundation and thereby

Art Unit: 3714

improve the gaming device by using an operating system customized for gaming using a scalable, free and widely supported operating system.

In regards to claim 8, *Bunnell* additionally teaches modifying the kernel to support specialized applications (such as gaming). *See col. 13:34-43*.

Claims 9, 10, 17, 38, 44 and 47 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of *Pascal*, as applied to the claims 22, 23 above, in further view of *Bock*, U.S. Patent 5,155,856 (Oct. 13, 1992) and *Davis*, U.S. Patent 6,401,208 (Jun. 4, 2002).

Bunnell discloses a scalable, modular operating system that selectively initializes only the parts of the kernel required under a system profile and does not attempt to initialize components that are not present. *See col. 11:20-28, 12:17-23, 13:3-12*. As a result, the system advantageously allows the operating system to configure kernel components in or out of the system. *See id.* Hence, *Bunnell* describes a system that disables selected device handlers.

Bunnell additionally discloses that the operating system can incorporate user-level code on several levels. For example, in a fully stand-alone embodiment, where only a single user-level application is executed, the application may be coupled directly with the operating system to allocate all the system resources to executing the single application. *See col. 8:39-46*. Accordingly, *Bunnell* does not limit what applications the developer includes within the kernel. *See 8:47-56*. Hence, *Bunnell* suggest coupling a user-level application with the operating system that is executed upon bootstrapping the system.

Pascal describes the process of executing an operating system from ROM upon bootstrapping a gaming device from a cold or warm start. *See fig. 2; col. 3:41-67*.

Art Unit: 3714

Thus, the operating system for a gaming device suggested by the combination of *Bunnell* with *Pascal* describes all the features of the present claims except zeroing-out unused RAM and testing and/or hashing the kernel. Regardless of the deficiencies, these features were known in the art at the time of the invention and would have been obvious to an artisan in view of *Bock* and *Davis*.

Bock describes “self-guarding” computer system that, upon initialization, zeros-out unused memory to erase information which is not required for subsequent operation. *See col. 1:38-62*. Erasing the data serves to eliminate vestigial data remaining in memory after reset that may interfere with the newly executed instructions and cause errors. *See col. 3:10-20*. Hence *Bock* suggests enhancing the security of a computing system by zeroing-out unused memory.

Davis describes a computing system that, upon initialization, tests and/or hashes the BIOS to enhance to system’s security. The reference describes several threats to computing device that circumvent security measures implemented at higher levels of operation. For example, a malefactor may replace the device’s ROM or a software virus may infiltrate the BIOS. In solution of this threat, *Davis* suggests testing the BIOS using a hash-function upon initialization. *See col. 1:32-2:5*.

The above references analogously describe means for initializing and increasing the security of an operating system. In view of *Bock* and *Davis*, it would have been obvious to a gaming artisan at the time of the invention to modify the gaming device operating system suggested by the combination of *Bunnell* with *Pascal* to add the features of zeroing-out unused RAM and testing and/or hashing the kernel. The modification would enhance the security of the gaming device by frustrating attempts to tamper with the operating system by modifying the data stored within the device’s memory prior to initialization.

In regards to claim 10, *Bunnell* additionally teaches making objects modular. *See col. 2:66-4:2*.

Claims 45, 46, 50 and 51 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of *Wiltshire*, U.S. Patent 6,409,602 (Jun. 25, 2002).

Bunnell teaches the operating system is scalable to be network compliant. *See col. 2:27-3:3*. As describes above, the reference describes all the features of the claimed subject matter except using the operating system to control a networked online system (*claim 45*) and using the system to control a progressive meter (*claim 46*). Regardless of the deficiencies, the features were known in the art at the time of the invention and would have been obvious to an artisan in view of *Wiltshire*.

Wiltshire discloses an analogous gaming system in which PC-based machines execute commercially available operating systems to decrease the cost of developing and upgrading gaming devices. *See col. 2:6-44, 4:44-7:5*. The reference describes an operating system controlling a networked, online gaming system including managing a progressive jackpot. *See col. 4:66-5:13, 5:45-64*.

In view of *Wiltshire*, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the gaming operating system suggested by *Bunnell*, wherein the operating system is scalable to support networks, to add the features of controlling a networked online system and controlling a progressive meter. The modification would allow the gaming device to communicate over a casino networks and thereby make the device attractive to operators whose casinos link gaming machines to networks for player tracking, accounting, monitoring, and linking to progressive jackpots.

Claims 4, 5, 38, 55-57 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of *Arbaugh et al.*, U.S. Patent 6,185,678 B1 (Feb 6, 2001).

As describe above, *Bunnell* teaches all the features of the claims except having the operating system verify that the operating system kernel or code for a shared object has not changed. Regardless of the deficiency, this feature would have been obvious to an artisan in view of *Arbaugh*.

Art Unit: 3714

Arbaugh discloses an architecture for initializing a computer system that ensures the integrity of the bootstrap process and provides reliability. *See col. 4:33-59*. Integrity is validated at each layer transition in the bootstrap process and a recovery process is included for integrity check failures. Ensuring the integrity is provided by the use of public key cryptography, a cryptographic hash function, and public key certificates. *See id.* The system does this by constructing a chain of integrity checks, beginning at power-on and continuing until the final transfer of control from the bootstrap components to the operating system itself. *See id.* The integrity checks compare a computed cryptographic hash value with a stored digital signature associated with each component. *See id.* Cryptographic algorithms are combined with the chosen protocols to add security to the recovery process, however if security is not a concern, then a less robust approach could be used. *See id.* Moreover, *Arbaugh* provides a useful summary of techniques for ensuring data integrity in data processing systems. In particular, the background teaches that it is known to verify software upon boot-up and while the operating system is running. *See col. 2:52-3:3*. By ensuring the integrity of the system, *Augaugh* teaches that security, integrity, reliability and total ownership cost will be improved. *See col 4:33-37, 4:60-65*.

In view of *Arbaugh*, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify the operating system described by *Bunnell*, wherein the operating system is customizable to support specialized application such as gaming, to add the feature of having the operating system verify that the operating system kernel or code for a shared object has not changed to improve the data processing systems security, integrity, reliability and total ownership cost.

In regards to claim 5, the combination above describes all the claimed features except crating a list of pointers that associate variable names with data locations. Regardless of is notoriously well known in the art of data processing to create a list of pointers that associate variable names with data locations such that a process can locate data by referencing variables. Thus, it would have been obvious to an

Art Unit: 3714

artisan at the time of the invention to add the feature of creating a list of pointers that associate variable names with data locations such that a process can locate data by referencing variables and thereby make programming simpler by not having to specifically reference memory addresses.

Claim 6 is are rejected under 35 U.S.C. 103(a) as being unpatentable over *Bunnell* in view of *Arbaugh*, as applied to claim 4 above, in further view of *Pascal*.

The combination of *Bunnell* with *Arbaugh* describes all the features of the claim except causing the execution of a corresponding callback function when a data variable is changed in non-volatile storage. Regardless of the deficiency, this feature would have been obvious to an artisan in view of *Pascal*.

Pascal discloses an analogous operating system for a gaming device wherein callbacks are employed to communicate information between application modules upon the occurrence of certain events. *See 1:44-2:30*. In generally, callback routines are used in state-based machines to communicate data between independent modules upon the occurrence of predetermined events. *See col. 6:25-45*. *Pascal* describes using callback to enhance to robustness of a gaming device under fault conditions to protect data that may affect the outcome of a game payout. *See col. 2:25-30*.

In view of *Pascal*, it would have been obvious to an artisan at the time of the invention to modify the customized gaming operating system suggested by the combination of *Bunnell* with *Fullerton*, wherein the system enhance security by monitoring application modules, to execute a callback function corresponding to a change in game data stored in non-volatile memory to enhance the security of the gaming device by monitoring changes in data that might affect the outcome of the game payout and thereby provide a more secure gaming device that is resistant to errors caused by losses in power or tampering.

Response to Arguments

Applicant's arguments filed December 9, 2002 have been fully considered but they are not persuasive.

First, the applicant asserts that the examiner fails to appreciate the complexity of using a system handler in a gaming system. The examiner disagrees. A system handler is an abstract device that automates the exchange of data between system levels and thereby reduces the effort needed to write applications. The concept of a system handler is well understood. Certainly, because gaming manufacturers employ a variety outdated and proprietary systems, writing software for a gaming system handler would be very difficult. However, a system is not patentable merely because it is complex. Instead, the standard of patentability is what would be obvious to one of ordinary skill in the art. The standard of patentability is what the prior art taken as a whole at a time prior to the invention suggests to an artisan. In this case, when adapting a operating system for use in a gaming system, it would be obvious to an artisan to employ a system handler to simplify the process of exchanging data between an processes and system components and thereby reduce the difficulty in developing software for the system.

Second, the applicant continues to argue that the claimed system distinguishes from the prior art because *Bunnell* is not analogous to a gaming system. The examiner disagrees. Merely employing the word "gaming" within a claim does not cordon off the invention from the whole of human knowledge. The standard of patentability is what would be obvious to one of ordinary skill in the art at the time the invention was made. This standard does not limit an artisan's knowledge to the field of gaming systems. Instead, it has been held that a prior art reference must either be in the field of applicant's endeavor or, if

Art Unit: 3714

not, then be reasonably pertinent to the particular problem with which the applicant was concerned. See *In re Oetiker*, 977 F.2d 1443, 24 USPQ2d 1443 (Fed. Cir. 1992).

In this case, *Bunnell* is reasonably pertinent to the particular problem which the applicant is concerned.

Generally, a data processor is generic device that processes data without regard for the type of data is contained in a program. So long as a program is compatible, the processor will execute it regardless of the type of data contained in the program. Hence, it is irrelevant whether the data represents gaming data. Therefore, claiming a "game controller", "game programs", and "game objects" are indistinguishable from claiming a "controller", "program" and "objects".

In specific regards to the claimed invention, the applicant contends that *Bunnell* is nonanalogous art because it does not describe a gaming system. The applicant identifies several problems to which the invention is directed. These include (i) lack of standardization between gaming devices due, in part, to reliance on primitive operating systems (*see p. 4, lines 16-21*); (ii) lack of uniformity in hardware interfaces (*see p. 5, lines 3-14*). To solve these problems, the applicant provides a "universal controller" tailored for use in gaming devices to improve the efficiency, security and operational costs.

In response, *Bunnell* discloses a "universal" operating system that can be scaled and reconfigured to support specialized computing applications and thereby reduce the time required to retrofit or modernize an existing operating system. *See col. 1:7-38, 2:59-64*. Thus, *Bunnell* is reasonably pertinent to the problem of providing an "universal controller" wherein the operating system for the specialized application of gaming.

Third, the applicant argues that Claim 1 is distinct from *Bunnell* because the reference (i) does not dynamically link program objects with a system handler and (ii) does not perform the operation at the kernel-level of execution. The examiner disagrees. In particular, claim 1 states "... a system handler

Art Unit: 3714

application operable to dynamically link with at least one gaming program object”. In response, *Bunnell* teaches an operating system that dynamically links a system handler and program objects into the kernel upon boot-up. *See col. 3:25-58, 4:47-5:25, 6:25-54, 8:57-63*. The applicant’s arguments admit the *Bunnell* dynamically links system objects, but contends that this linking is different than claimed. *See amendment D (paper no. 16), p. 10, ¶ 2*. The examiner disagrees. *Bunnell* teaches dynamically linking system objects that, in an embedded system (such as a gaming system), may include program objects. *See 8:57-63*. Hence, *Bunnell* describes the claimed feature of dynamically linking. Furthermore, in response to the applicant’s assertion that the embedding of POSIX functions within the kernel is “a general reference with no probative teaching”, the assertion is baseless. The teaching is sufficiently supported to communicate the feature to an artisan. *See col. 7:62-8:36*. Thus, in regards to the feature of dynamic linking, the rejection is maintained.

Fourth, under the heading of “Effects of the Amendments establishing Non-Obviousness”, the applicant states that the claimed invention distinguishes from *Bunnell* because the reference calls an API directly and selects programs to be executed out of the API. However, the applicant’s arguments do not comply with 37 CFR 1.111(c) because they do not clearly point out the patentable novelty which he or she thinks the claims present in view of the state of the art disclosed by the references cited or the objections made. Further, they do not show how the amendments avoid such references or objections. Notably, the primary purpose of an API is to provide a layer of abstraction to dynamically link program objects in and out of the system. In comparison, *Bunnell* teaches that in an embedded system (such as a gaming system), the API processes can be eliminated by executing program objects at the kernel level and thereby reduce system overhead and resource requirements needed to support a higher level of abstraction. *See col. 8:58-63*.

Art Unit: 3714

Fifth, the applicant contends that the prior art does not teach dynamic unlinking. The examiner disagrees. *Bunnell* describes a customizable UNIX-based operating system that dynamically links operating system components upon boot-up and thereby provide a scalable and efficient operating system for supporting specialized computing environments (such as gaming systems). See col. 1:26-2:64, 13:34-42. Additionally, *Bunnell* teaches system objects in an embedded system (such as a gaming system) may include program objects. See 8:57-63. Furthermore, the reference suggests that the methods described may be applied in operating system other than UNIX. See col. 13:34-42. However, *Bunnell* does not describe dynamic unlinking wherein kernel objects are removed after processing has begun. Regardless, this feature is known in the art.

Rusling describes the LINUX operating system. LINUX allows program objects to be dynamically loaded and unloaded as they are needed after the system has booted. See chapter 12. Dynamic loading is beneficial because it keeps the kernel size to a minimum and makes the kernel very flexible. See *id.* For example, in “demand loading” objects are loaded into the kernel only as needed. See *id.* Typically, kernel objects are stored in a library of objects and are linked just like any other programs in the system. See *id.*

Thus, the feature of dynamically loading and unloading program objects is known in the art.

Sixth, the applicant generally argues that, although it is possible to use modern PC's and operating systems to write games, it was neither taught nor suggested in the prior art to modify PC's to meet gaming regulations. The examiner disagrees. It is known in the art to use PC's and operating systems within gaming systems. See, e.g., *Wiltshire*, col. 4:49-7:6. Furthermore, commercial-off-the-shelf (COTS) processing systems that use PC processors with operating systems such as LINUX or UNIX for industrial applications (e.g. gaming) are well known. Moreover, the applicant's contention that it would not be obvious to an artisan to modify a PC-based system to meet gaming regulation is

Art Unit: 3714

unpersuasive. An artisan using an embedded-PC in a gaming system would be highly motivated to make the modifications in order to meet regulatory requirements. Notably, even though it has not been common practice in the gaming industry to use PC-based systems for gaming, this is not the standard for patentability. The standard of patentability is what the prior art taken as a whole at a time prior to the invention suggests to an artisan. In demonstrating the knowledge of an artisan, it has been held that a prior art reference must either be in the field of applicant's endeavor or, if not, then be reasonably pertinent to the particular problem with which the applicant was concerned. See *In re Oetiker*, 977 F.2d 1443, 24 USPQ2d 1443 (Fed. Cir. 1992). In this case, one of ordinary skill in the art would possess knowledge of embedded-PC's and it would have been obvious to modify an embedded-PC used in a gaming device to meet the gaming regulations.

Seventh, the applicant generally argues that the prior art does not recognize the benefits of dynamic linking. The examiner disagrees. *Rusling* teaches that the LINUX operating system allows program objects to be dynamically loaded and unloaded as they are needed after the system has booted to keep the kernel size to a minimum and make the kernel very flexible. See *chapter 12*.

Consequently, for all the reasons given above, the rejection is maintained.

Conclusion

The following prior art of record is not relied upon but is considered pertinent to applicant's disclosure:

Michael Tiemann, 'Why Embedded Linux' <<http://linuxdevices.com/cgi-bin/printerfriendly.cgi?id=AT8926600504>> (Oct. 28, 1999) describes the benefits of using a LINUX-based operating system using pc-based hardware in a embedded system (such as a gaming machine).

Art Unit: 3714


Rick Lehrbaum, 'Why Linux' <<http://linuxdevices.com/cgi-bin/printerfriendly.cgi?id=AT9663974466>> (Jan. 21, 2000) describes the benefits of using a LINUX-based operating system using pc-based hardware in a embedded system (such as a gaming machine).

Rick Lehrbaum, 'Why Linux' <<http://linuxdevices.com/cgi-bin/printerfriendly.cgi?id=AT3611822672>> (Feb. 19, 2000) describes the benefits of using a LINUX-based operating system using pc-based hardware in a embedded system (such as a gaming machine).

U.S. Patent 5,995,745 To Yodaiken describes a method for modifying the LINUX operating system to support real-time operations.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Steven Ashburn whose telephone number is 703 305 3543. The examiner can normally be reached on Monday thru Friday, 8:00 AM to 4:30 PM. If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tom Hughes can be reached on 703-308-1806. The fax phone numbers for the organization where this application or proceeding is assigned are 703 872 9302 for regular communications and 703 872 9303 for After Final communications. Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703 308 1078.

S.A.
March 13, 2003



MARK SAGER
PRIMARY EXAMINER